Problem Set 3: Due by October 30, 2006

Problem 1. (40 points)

a) Suppose we have a set of n elements. Suppose we generate random permutations as follows. Show that R1 randomly permutes A or show that it is not correct.

```
R1(A[1..n],n)
for(i=1;i<=n;i++) {
  choose j randomly from 1 to n inclusive;  // there is an 1 (one) here
  swap A[i] and A[j] ;
}
```

b) Suppose we slightly modify this algorithm as follows Show that R2 randomly permutes A or show that it is not correct.

```
R2(A[1..n],n)
for(i=1;i<=n;i++) {
  choose j randomly from i to n inclusive ; // there is an i (eye) here
  swap A[i] and A[j] ;
}
```

Problem 2. (40 points)

This continues problem 1. The `Highlander Casino Corporation` plays a game of random permutations on 4 elements as follows.

- The client bets on a permutation of his/her choice.
- The casino uses the incorrect one of the algorithms to generate a random permutation.
- The casino pays the client the fair amount of money if the client won (i.e. guessed right). Fair money means in the long run neither the client nor the casino would make any money IF their algorithm chose correctly and fairly a permutation (i.e. for n elements the casino pays n dollars for every dollar bet right).

Since the casino's algorithm is incorrect, what methodology could you follow to gain over the casino?
What edge (percentage-wise) do you expect to gain over the casino? Explain.

Problem 3. (40 points)

You are given n polynomials $f_1(x), \ldots, f_n(x)$ each one of degree $n-1$, i.e. each polynomial looks similar to the generic polynomial $f(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. You are also given n points $c_1, c_2, \ldots, c_n$. Give an algorithm that evaluates the n polynomials in the n points in $O(n^{\lg 7})$ time.

Problem 4. (40 points)

Evaluate the following polynomial of degree n, at $x = c$ using $o(n)$ additions/subtractions and $o(n)$ multiplications (you are not allowed to use divisions or any operations other than $+, -, *$). **Hint:** What do you know about the right hand side of $f(x)$?

$$f(x) = \sum_{k=0}^{n} \binom{n}{k} x^k$$

[Note: $f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$; for other definitions see Problem 5.]

Problem 5. (40 points)

n choose k. $\binom{n}{k}$ is defined below with $k!$ defined as $k! = k(k-1)(k-2)\ldots 1$.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Find $\binom{n}{k}$ in $O(n^2)$ additions/subtractions. Multiplications and divisions are not allowed.

If you are not very familiar with the properties of $\binom{n}{k}$, a good reading is section C.1 or 6.1 (new or older version of the textbook) including the exercises. A way to derive the answer is hidden somewhere there.

Problem 6. (50 points)

Solve the recurrence for the number of operations in Strassen's method exactly, i.e. solve

$$M(n) = 7M(n/2) + (18/4)n^2.$$

What is the base case? Explain. How many operations are required for the base case? Explain. Does it matter whether one chooses the base case to be $n = 1$ or $n = 2$?

What happens for the questions above if one uses the modification of the textbook with the recurrence becoming

$$M(n) = 7M(n/2) + (15/4)n^2.$$

Programming

Problem P1. (120 points)

Strassen.

```
// n can be any integer dimension ; i.e. you have to take care and make it
//   a power of two if necessary
// *A, *B, *C are one dimensional arrays of n*n elements (floats)
// A[j*n+i] is the i-th row and j-th column element of a two dimensional array
// For Java use one dimensional arrays
Strassen(float *A, float *B, float *C, int n)  //Does Strassen for arbitrary n
ReadMatrix(float **A,int *n, file input-file); //Allocates space for A and reads A
SetMatrix(float *A,float *B,int n); //Allocates space for A and reads A
PrintMatrix(float *A,int n,  file output-file,)//Prints A into file output-file
```

You need to implement the following interface

```
% ./strassen  input-A input-B output-C
or
% java strassen input-A input-B output-C
```

where `input-A, input-B` are files containing input matrices A, B and `output-C` is the file that will contain the output $A \times B$ of Strassen's method. All three files have the same format. The first line contains the dimension n in the form of an integer. Subsequent lines contain in the form of floats the input elements in row-major format. That is the first $n = 5$ values `1.0 2.0 3.0 4.0 5.0` are the elements of the first row of the 5×5 array. The next 5 values `6.0 7.0 8.0 9.0 and 10.0` are the elements of the second row and so on. Files `input-A, input-B` are read through `ReadMatrix` and file `output-C` is written by `PrintMatrix`. `SetMatrix` allows one to set copy B into A internally.

```
5
1.0 2.0 3.0 4.0 5.0
6.0 7.0
     8.0 9.0 10.0
1.0
  2.0 3.0 4.0 5.0
1.0
  2.0 3.0 4.0 5.0
1.0 2.0 3.0 4.0 5.0
```

Note. The input array(s) can be of dimension say 17×17. After reading such matrices it's up to you to decide how to store such a matrix; Strassen (textbook description) can only deal with 16×16 or 32×32 matrices but not a 17×17 one. When you print the results into `output-C` make sure it is that of a 17×17 matrix and not that of a matrix of some other dimension. For other assumptions, deviations or instructions, provide a `readme.txt` file with your code.

Problem P2. (40 + 60 = 100 points)

This Problem can only be done in C or C++ because of the interface of the `qsort` function. If you do it you get a maximum of 40 points and you enter a contest with other fellow students. The only one winner (time-wise) receives 60 bonus points. If there is only one submission that works the winner takes all. Benchmark or final-code testing programs won't be divulged. Debugging code is available through the CS 435 web-page at

`http://www.cs.njit.edu/ alexg/courses/cs435/handouts.html` section B4.

Implement a randomized quick-sort `ranqsort` function with the interface of the Standard C library qsort function. For the syntax of qsort do `man qsort` on afs. The interface of your quick-sort would thus be.

```
void qsort(void *base, size_t nel, size_t width,int (*compar)(const void *, const void *));
```

Your implementation must thus be called with the following interface.

```
void ranqsort(void *base, size_t nel, size_t width,int (*compar)(const void *, const void *));
```

A deviation from this interface will gain you 0 points.